Software Testing and Verification

Problem Set 1: Black-Box Testing

Apply the black-box test case design techniques discussed in class to the two program specifications given below. (Both specifications are simplified versions of actual MAN pages.)

Deliverables for each of the two functions are:

1. A lists of appropriate Causes and Effects together with an unambiguous description of each;

2. One or more Cause-Effect graphs reflecting relationships and constraints deducible from the specification;

3. A test case coverage matrix reflecting all combinations of connected Causes resulting in each Effect as represented in the graph(s);

4. A list of specific Boundary Values that you feel should be covered by additional test cases (do not generate the test cases, just DESCRIBE the boundary values);

5. A list of ADDITIONAL specific conditions or usage scenarios that you feel would be worthwhile to cover by applying Intuition & Experience.

Note: Questions re ambiguities or other potential requirements issues would normally be directed to the specification author(s). However, since none are available, please make and document any (reasonable) assumptions that are necessary for you to complete this assignment.

NAME

fmod - floating-point remainder value function

SYNOPSIS

fmod(x, y);

DESCRIPTION

The fmod() function returns the floating-point remainder of the division of x by y.

RETURN VALUES

The fmod() function returns the value x - i * y, for some integer i such that, if y is non-zero, the result has the same sign as x and magnitude less than the magnitude of y.

If x or y is NaN, NaN is returned. If y is 0, NaN is returned (and errno is set to EDOM by matherr). If x is +/-Inf, NaN is returned. If y is non-zero, fmod(+/-0, y) returns the value of x. If x is not +/-Inf, fmod(x, +/-Inf) returns the value of x.

USAGE AND ERRORS

An application wishing to check for error situations should set errno to 0 before calling fmod(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

NAME

sort - sort, merge, or sequence check text files

SYNOPSIS

sort [-bcdfimMru] [-o output] [file ...]

DESCRIPTION

The sort command sorts lines of all the named files together and writes the result on the standard output.

Comparisons are based on the entire input line. Lines are ordered according to the collating sequence of the current locale.

OPTIONS

The following **options** alter the default behavior:

-c Checks that the single input file is ordered as specified by the arguments and the collating sequence of the current locale. The exit code is set and no output is produced unless the file is out of sort.

-m Merges only. The input files are assumed to be already sorted.

-u Unique: suppresses all but one in each set of equal lines.

-o output

Specifies the name of an output file to be used instead of the standard output. This file can be the same as one of the input files.

Ordering Options

The default sort order depends on the value of environment variable LC_COLLATE. If LC_COLLATE is set to C, sorting will be in ASCII order. If LC_COLLATE is set to en_US, sorting is case insensitive.

The following options override the default ordering rules.

-d "Dictionary'' order: only letters, digits, and blanks (spaces and tabs) are significant in comparisons.

-f Folds lower-case letters into upper case.

-i Ignores non-printable characters.

-M Compares as months. The first three non-blank characters of each line are folded to upper case and compared. For example, in English the sorting order is "JAN" < "FEB" < ... < "DEC". Invalid characters compare low to "JAN". The -M option implies the -b option (see below).

-r Reverses the sense of comparisons.

-b Ignores leading blank characters in a line.

OPERANDS

The following operand is supported:

file A path name of a file to be sorted, merged or checked. If no file operands are specified, or if a file operand is -, the standard input will be used.

EXAMPLES

Example 1: Sorting in "dictionary" order with lower case folding

The following command sorts the contents of infile into "dictionary" order without regard to case:

```
example% sort -df infile
```

Example 2: Sorting in reverse order

The following command sorts, in reverse order, the contents of infile1 and infile2, placing the output in outfile:

```
example% sort -r -o outfile infile1 infile2
```

Example 3: Printing sorted lines excluding lines that duplicate a field

The following command prints the lines of the already sorted file infile, suppressing all but one occurrence of its lines:

```
example% sort -um infile
```

EXIT STATUS

The following exit values are returned:

0 All input files were output successfully, or -c was specified and the input file was correctly sorted.

1 Under the -c option, the file was not ordered as specified, or if the -c and -u options were both specified, two input lines were found to be equal.

>1 An error occurred.

DIAGNOSTICS

Comments and exits with non-zero status for various trouble conditions (for example, when input lines are too long), and for disorders discovered under the -c option.

NOTES

When the last line of an input file is missing a new-line character, sort appends one, prints a warning message, and continues.